Ausbildung Fachinformatiker Anwendungsentwicklung

Programmierübung (mit Musterlösung)

C/Python–Programmierung / Simulation

1 Aufgabe

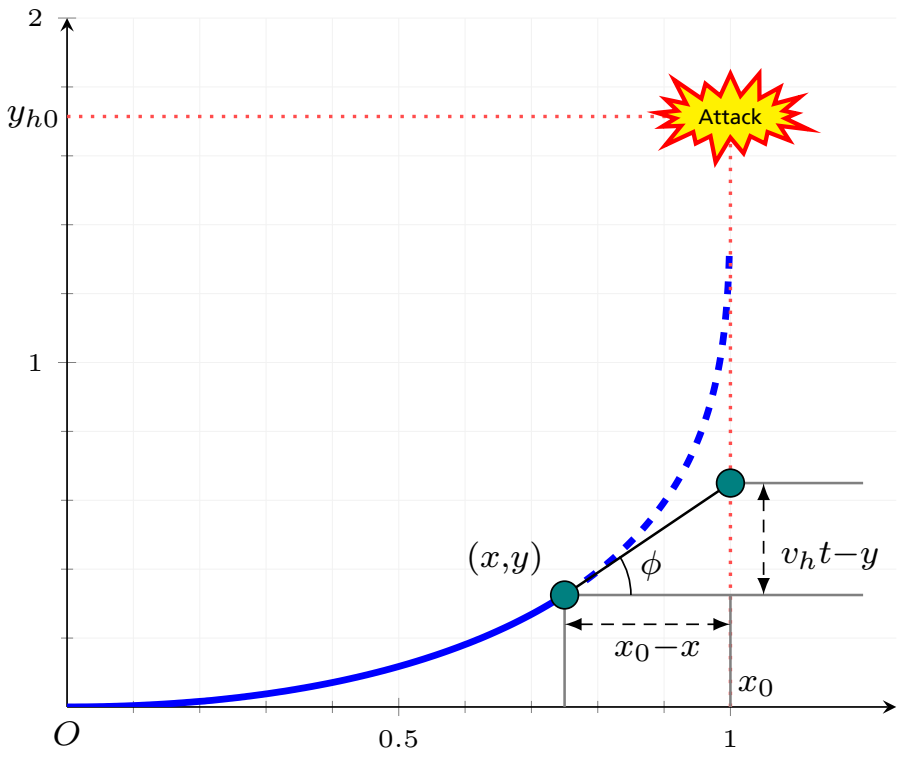
Es ist ein Programm (`pirate.exe` bzw. `pirate.py`) in der Programmiersprache C und der Programmiersprache `Python` zu schreiben, das folgendes Szenario lösen kann:

Ein Handelsschiff segelt mit konstanter Geschwindigkeit (v_h) in einem kartesischen Koordinatensystem vertikal in y–Richtung. Es befindet sich anfänglich an der Position $(x_0, 0)$ und sein Weg kann beschrieben werden durch die Kurve $x(t) = x_0$ und $y(t) = v_h t$.

Ein Piratenschiff an den Koordinaten $(0, 0)$ segelt zeitgleich mit dem Handelsschiff los und nimmt permanent Kurs auf die aktuelle Position des Handelsschiffs. Natürlich ist das Piratenschiff schneller, es hat die Geschwindigkeit $v > v_h$. D.h. $n = \frac{v_h}{v} < 1$. Wählt man also geeignete Einheiten, so kann man für die Berechnung setzen: $x_0 = x_h = 1$,$v = 1$ und $v_h = n < 1$ (Konkret zu Berechnung: $n = 3/4$).

Irgendwann wird das Piratenschiff das Handelsschiff eingeholt haben. Die Frage ist:

An welcher y–Position y_{h0} hat das Piratenschiff das Handelsschiff eingeholt. Da das Handelsschiff mit konstanter Geschwindigkeit segelt, ist die Frage natürlich äquivalent zu der Frage, zu welcher Zeit $t_{h0} = \frac{y_{h0}}{v_h}$.



Version vom 2024-12-29

2 Vorgehensweise

Simulieren Sie in kleinen Zeitschritten Δt die Bewegung beider Schiffe. Sie bekommen durch Wahl des geeigneten Zeitschritts ein hinreichend genaues Ergebnis.

Die beiden Schiffe treffen aufeinander, wenn der Abstand beider Schiffe sowohl in x–Richtung als auch in y–Richtung kleiner als ϵ ist. Dabei ist ϵ geeignet zu wählen. Die Wahl von ϵ und Δt sind voneinander abhängig, denken Sie darüber nach.

Zur Erleichterung wurde schon die notwendige Tabelle erstellt. Diese ist folgendermaßen von links nach rechts und oben nach unten zu lesen.

1. Für ein bestimmtes t (also auch k) werden die Position für das Handelsschiff $P_h = (x_h, y_h)$ und die Positionen des Piratenschiffes $P = (x, y)$ berechnet.
2. Es wird der (Kurs–)Winkel zwischen dem Handelsschiff und dem Piratenschiff aus den Abständen mit Hilfe des $\arctan()$ berechnet.
3. Mit dem Kurswinkel kann nun das jeweilige Δx und Δy berechnet werden, die das Piratenschiff im nächsten Zeitintervall vorwärts segelt.

k	t	x_h	y_h	x	y	$\tan\phi$	Δx	Δy
0	0	1	0	0	0	0	Δt	0
1	$1\Delta t$	1	$1n\Delta t$	$x^{(0)} + \Delta x^{(0)}$	$y^{(0)} + \Delta y^{(0)}$	$\frac{y_h^{(1)} - y^{(1)}}{1 - x^{(1)}}$	$\Delta t \cos\phi^{(1)}$	$\Delta t \sin\phi^{(1)}$
2	$2\Delta t$	1	$2n\Delta t$	$x^{(1)} + \Delta x^{(1)}$	$y^{(1)} + \Delta y^{(1)}$	$\frac{y_h^{(2)} - y^{(2)}}{1 - x^{(2)}}$	$\Delta t \cos\phi^{(2)}$	$\Delta t \sin\phi^{(2)}$
3	$3\Delta t$	1	$3n\Delta t$	$x^{(2)} + \Delta x^{(2)}$	$y^{(2)} + \Delta y^{(2)}$	$\frac{y_h^{(3)} - y^{(3)}}{1 - x^{(3)}}$	$\Delta t \cos\phi^{(3)}$	$\Delta t \sin\phi^{(3)}$
$\ldots$								
k_0	$k_0\Delta t$	1	$k_0 n\Delta t$	$x^{(k_0-1)} + \Delta x^{(k_0-1)}$	$y^{(k_0-1)} + \Delta y^{(k_0-1)}$	$\frac{y_h^{(k_0)} - y^{(k_0)}}{1 - x^{(k_0)}}$	$\Delta t \cos\phi^{(k_0)}$	$\Delta t \sin\phi^{(k_0)}$

Die Iteration kann beendet werden wenn

$$|x_h - x| < \epsilon \qquad \text{und} \qquad |y_h - y| < \epsilon$$

Die Zeit t_{h0} und die Entfernung y_{h0} ergeben sich dann aus den Werten $k_0\Delta t$ und $k_0 n\Delta t$.

Simulieren Sie dieses Problem sowohl mit der Programmiersprache `Python` als auch in der Programmiersprache `C`. Versuchen Sie verschiedene Werte von Δt und ϵ. Welche Beobachtungen machen Sie beim Testen?

3 Bewertung

Ziel ist ein möglichst exakter Wert des Ergebnisses. Beobachtungen während der Entwicklung und während des Testens sollen in einer kurzen Dokumentation niedergeschrieben werden (1–2 Seiten).

4 Abgabe und Zeitrahmen

Der Zeitrahmen dieser Aufgabe ist mit etwa 24 Stunden angesetzt.

5 Musterlösung

Das Problem wurde bekannt durch den Franzosen *Pierre Bouguer* (1698–1758), im deutschsprachigen Raum bekannt unter dem Namen *Verfolgungskurve* oder *Hundekurve*. Das Problem selbst tritt in vielen Zusammenhängen auf und wurde wahrscheinlich schon vor mehr als tausend Jahren im asiatischen Raum behandelt.

Das Problem hat eine analytisch präzise Lösung, die mit der iterierten Lösung gut verglichen werden kann.

$$y_{h0} = \frac{n}{1-n^2}, \qquad \text{wenn} \qquad x_0 = 1 \qquad \text{und} \qquad n = \frac{v_h}{v}$$

Für das in der Aufgabe genannte konkrete Beispiel findet sich:

$$n = \frac{3}{4} \qquad \Rightarrow \qquad y_{h0} = \frac{12}{7} = 1.\overline{714285}$$

Dies ist ein typisches Simulationsproblem (ähnlich der Wettervorhersage). Möchte man genaue Ergebnisse, muss man u.U. sehr lange auf das Ergebnis warten.

6 Prinzipielle Schwierigkeiten

Zum einen stellt man schon durch Überlegung fest, dass die zeitliche Schrittweite (Δt) immer kleiner oder gleich der angestrebten Genauigkeit (ϵ) sein muss. Andernfalls kann es nämlich passieren, dass mit einem Zeitschritt, das Piratenschiff das Handelsschiff überholen kann und dann die Abstände immer nur noch größer werden.

Das zweite Problem ist die Maschinengenauigkeit, die eine natürlich Grenze der Präzision darstellt.

Das dritte Problem ist die Verwendung des $\arctan()$, der bei den großen Werten, nahe bei $90°$ nur noch ungenaue Werte liefern kann (Man sehe sich die Kurven von $\tan()$ und $\arctan()$ dazu an). Die Kurve schmiegt sich immer dichter an eine Asymptote an, so dass Änderungen mit der Präzision nicht mehr darstellbar sind.

Das C–Programm ist sehr schnell und liefert eine Genauigkeit von etwa 10 Stellen bei einer Rechenzeit von ca. $2h$. Es wird die Maschinengenauigkeit benutzt. Allerdings wurde hier der Datentyp `long double` gewählt und alle Funktionen jeweils mit einer Variante für diesen Datentyp benutzt. `long double` ist nicht normiert und muss nur mindestens so lang wie `double` sein. Auf OSX ist der Typ 80 Bit groß (wird aber für einfacheren Zugriff in 128 Bit Worten abgespeichert, daher kann ein `sizeof(long double)` zwar 128 Bit ergeben, aber in der Dokumentation steht eine Länge von 80 Bit) und besitzt eine Mantisse von 64 Bit für die Präzision. Dabei ist darauf zu achten, dass *durchgängig* das `long double` Format benutzt wird, denn nur *eine* Funktion, die die Präzision verschlechtert, verschlechtert auch die gesamte Präzision.

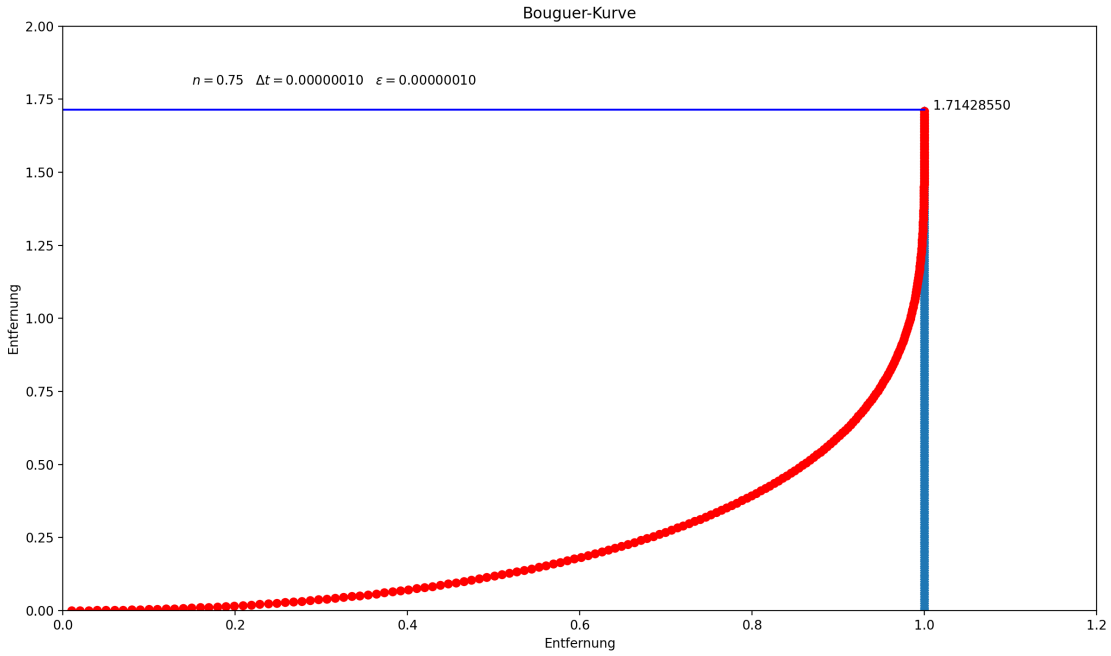
Die Python–Variante ist deutlich langsamer, da sie eine interpretierte Sprache ist.

Ein Wechsel auf eine Python–Bibliothek (`mpmath`) kann hier zwar eine bessere und theoretisch beliebige Genauigkeit erzielen, aber die Rechenzeit wächst mit zunehmend kleiner werdendem Δt beträchtlich[1].

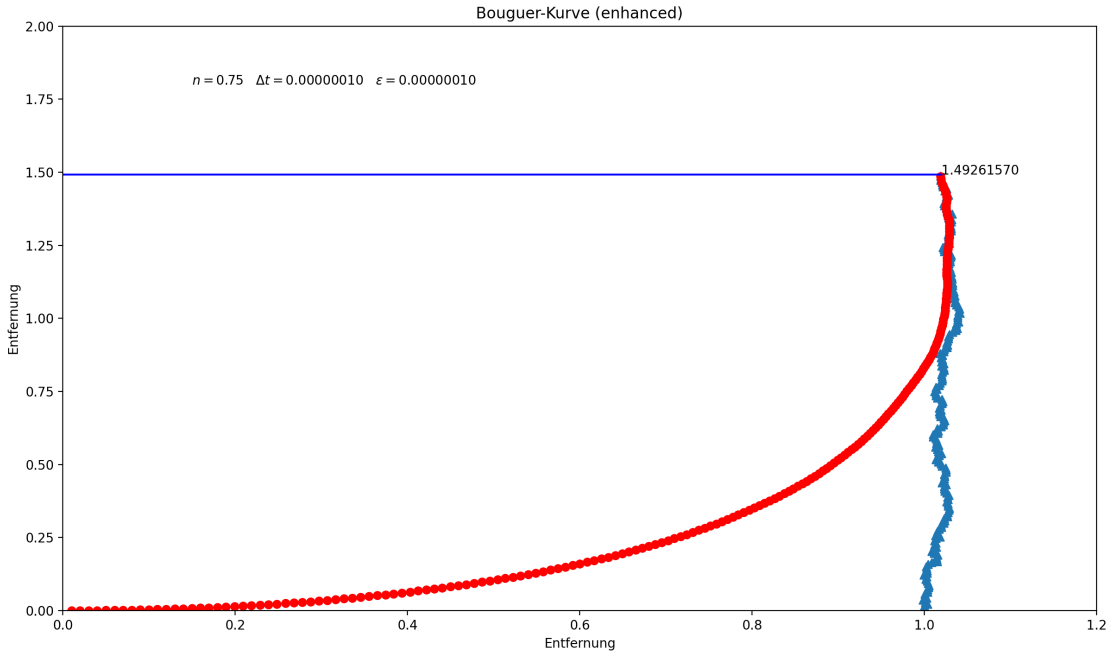
Es wird klar, dass z.B. die Wettervorhersage, die prinzipiell ähnliche Modelle benutzt um Vorhersagen zu erzeugen, für weitreichende Vorhersagen nicht nutzbar ist.

[1] Für C gibt es auch eine Bibliothek für größere Genauigkeit (`gmp`), aber da die Bibliothek mit dem Compiler unter Mac viele Fehler aufweist, wurde diese nicht getestet.

7 Die Programme



Grafische Ausgabe von `Aufgabe03Simple.py`



Grafische Ausgabe von `Aufgabe03Enhanced.py`

Musterlösung für `pirate.c`

```c
/**
 * Program simulates the pirate problem (long double version)
 * Author: Frank Zimmermann
 * Date:  28.10.2020
 *
 * Genauigkeit:
 * delta t = 1 * 10^-10
 * eps     = 1 * 10^-10
 * Ergebnis: 10 Digits
 * Dauer ca. 7000 s ; ca 2 Stunden
 */
#include <stdio.h>
#include <math.h>
#include <time.h>

int main(int argc, char** argv) {
        clock_t prgstart, prgende;
        long double dt = powl(10,-10);
        long double eps = powl(10,-10);
        long double n = 3. / 4.;
        long double dx = dt;
        long double xh = 1;
        long double t = 0, yh = 0, x = 0, y = 0, k = 0, phi = 0, dy = 0;
        prgstart=clock();
        printf("k=%11.0Lf      xh-x=%15.13Lf    yh-y=%15.13Lf\n", k,(xh -x), (yh -y));
        while (( (long double)(yh - y) > eps || (long double)(xh - x) > eps ) && k < powl(10,12)){
                k = k + 1;
                t = t + dt;
                yh = k * n * dt;
                x = x + dx;
                y = y + dy;
                /* to avoid division by zero */
                if (x != 1) {
                        phi = atanl((yh - y) / (1 - x));
                } else {
                        phi = M_PI / 2;
                }
                dx = dt * cosl(phi);
                dy = dt * sinl(phi);
                if (fmodl(k, powl(10,8)) == 0) {
                        printf("k=%11.0Lf      xh-x=%15.13Lf    yh-y=%15.13Lf\n", k,(xh -x), (yh -y));
                }
        }
        printf("%15.13Lf/%15.13Lf    k0=%Lf      k*dt*n=%15.13Lf      n/(1-n^2)=%15.13Lf\n",
               dt, eps, k, k * dt * n, n / (1 - n * n));
        printf("xh-x=%15.13Lf    yh-y=%15.13Lf\n", (xh -x), (yh-y));
        prgende=clock();
        printf("Die Programmlaufzeit betrug %.3f Sekunden\n",
      (float)(prgende-prgstart) / CLOCKS_PER_SEC);
        return 0;
}
```

Musterlösung für `Aufgabe03Simple.py`

```
# pirate problem
# Frank Zimmermann
# 2.11.2020
import time
import math
import matplotlib.pyplot as plt
import numpy as np

# Setzen der Anfangsparameter
dt = 0.0000001
eps = 0.0000001
n = 3 / 4

# Für graphische Ausgabe
plt.ion()
fig, ax = plt.subplots()
xp, yp = [], []
xph, yph = [], []
sc1 = ax.scatter(xph, yph, marker='^')
sc2 = ax.scatter(xp, yp, c="red")
plt.xlim(0, 1.2)
plt.ylim(0, round(n / (1 - n ** 2) + 0.5))
plt.title('Bouguer-Kurve')
fig.canvas.manager.set_window_title('SVLFG␣Aufgabe␣03')
plt.text(0.15, 0.9 * round(n / (1 - n ** 2) + 0.5), r'$n=$' + '{:3.2f}␣␣␣'.format(n) +
        r'$\Delta␣t=$' + '{:9.8f}␣␣␣'.format(dt) + r'$\epsilon=$' + '{:9.8f}'.format(eps))
plt.ylabel('Entfernung')
plt.xlabel('Entfernung')
# plt.grid(True)
plt.draw()
#################################

start = time.time()
t = 0.
xh = 1.
yh = 0.
x = 0.
y = 0.
xx = 0.
yy = 0.
k = 0.
phi = 0.
dx = dt
dxx = dt
dy = 0.
dyy = 0.
while (math.fabs(yy - yh) > eps or math.fabs(xx - xh) > eps) and k < 10000000000:
    k = k + 1
    t = t + dt
    yh = k * n * dt
    x = x + dx
    y = y + dy
    xx = xx + dxx
    yy = yy + dyy
    # Variante 1
    # Trigonometrie
    if x != 1:
        phi = math.atan((yh - y) / (1 - x))
```

```
    else:
        phi = math.pi / 2
    dx = dt * math.cos(phi)
    dy = dt * math.sin(phi)
    # Variante 2
    # Pythagoras
    dxx = dt * (1 - xx) / math.sqrt((yh - yy) ** 2 + (1 - xx) ** 2)
    dyy = dt * (yh - yy) / math.sqrt((yh - yy) ** 2 + (1 - xx) ** 2)
    if math.fmod(k, 100000) == 0:
        # print("phi=", '{:20.18f}'.format(phi), " xh-x =",
        #       '{:20.18f}'.format(math.fabs(x - xh)),
        #       "    yh-y =", '{:20.18f}'.format(math.fabs(y - yh)))
        # print("Hyp=", '{:20.18f}'.format(math.sqrt((yh-yy)**2+(1-xx)**2)) ,
        #       "     xh-xx =", '{:20.18f}'.format(math.fabs(xx - xh)),
        #       "   yh-yy =", '{:20.18f}'.format(math.fabs(yy - yh)))
        # Graphische Ausgabe
        xp.append(xx)
        yp.append(yy)
        xph.append(xh)
        yph.append(yh)
        sc1.set_offsets(np.c_[xph, yph])
        sc2.set_offsets(np.c_[xp, yp])
        fig.canvas.draw_idle()
        plt.pause(0.001)
distance = k * dt * n
plt.plot([0, 1.0], [distance, distance], 'b')
plt.text(1.01, distance, '{:9.8f}'.format(distance))

#plt.draw()
fig.canvas.draw_idle()
########################
print(dt, "/", eps, ":", "k0=", k, distance, n / (1 - n ** 2))
print("xh-x=", math.fabs(x - xh), "␣␣␣yh-y=", math.fabs(y - yh), "␣␣eps=", eps)
ende = time.time()
print('{:5.3f}s'.format(ende - start))
plt.waitforbuttonpress()
```

Musterlösung für `Aufgabe03.py`

```
# pirate problem with extended precision
# Frank Zimmermann
# 2.11.2020
import time
from mpmath import *

start = time.time()
mp.dps = 20
print(mp)
dt  = mp.mpf('0.0000010000000')
eps = mp.mpf('0.0000010000000')
print(nstr(eps, 20))
n = mp.mpf('0.75')
t = mp.mpf('0')
xh = mp.mpf('1')
yh = mp.mpf('0')
# x = mp.mpf('0')
# y = mp.mpf('0')
k = mp.mpf('0')
```

```
alpha = mp.mpf('0')
# dx = mp.mpf(dt)
# dy = mp.mpf('0')
xx = mp.mpf('0')
yy = mp.mpf('0')
dxx = mp.mpf(dt)
dyy = mp.mpf('0')
while (mp.fabs(mp.fsub(yy, yh)) > eps) or (mp.fabs(mp.fsub(xx, xh)) > eps) and k < 1000000000:
    k = mp.fadd(k, 1)
    t = mp.fadd(t, dt)
    yh = mp.fmul(k, mp.fmul(n, dt))
    # x = mp.fadd(x, dx)
    # y = mp.fadd(y, dy)
    # if x != 1:
    #     alpha = mp.atan(mp.fsub(yh, y) / mp.fsub(1, x))
    # else:
    #     alpha = pi/2
    # dx = mp.fmul(dt, mp.cos(alpha))
    # dy = mp.fmul(dt, mp.sin(alpha))
    xx = mp.fadd(xx, dxx)
    yy = mp.fadd(yy, dyy)
    dxx = mp.fmul(dt,mp.fsub(1.,xx)/(mp.sqrt(mp.fadd(mp.power(mp.fsub(1.,xx),2),
                                                     mp.power(mp.fsub(yh,yy),2)))))
    dyy = mp.fmul(dt,mp.fsub(yh,yy)/(mp.sqrt(mp.fadd(mp.power(mp.fsub(1.,xx),2),
                                                     mp.power(mp.fsub(yh,yy),2)))))
    # if mp.fmod(k, 100000) == 0:
    #     print("xh-x=", mp.fabs(mp.fsub(x, xh)), " yh-y=", mp.fabs(mp.fsub(y, yh)))
    # if mp.fmod(k, 1000000) == 0:
    #     print("k=", k, "k*dt*n=", nstr(k * dt * n, 20), "x,y=", nstr(x, 20), nstr(y, 20))
print(nstr(dt, 20), "/", nstr(eps, 20), ":", "k0=", k, nstr(k * dt * n, 20),
      nstr(n / (1 - n ** 2), 20))
# print("xh-x=", nstr(mp.fabs(mp.fsub(x, xh)), 20), " yh-y=",
#       nstr(mp.fabs(mp.fsub(y, yh)), 20), " eps=", nstr(eps, 20))
print("xh-xx=", nstr(mp.fabs(mp.fsub(xx, xh)), 20), "␣␣␣yh-yy=",
      nstr(mp.fabs(mp.fsub(yy, yh)), 20), "␣␣eps=", nstr(eps, 20))
stop = time.time()
print('{:5.3f}s'.format(stop - start))
```

Musterlösung für `Aufgabe03Enhanced.py`

```
# pirate problem
# Frank Zimmermann
# 2.11.2020
import time
import math
import matplotlib.pyplot as plt
import numpy as np
import random

random.seed()
mu = math.pi/2 # Hauptbewegungsrichtung nach Norden
kappa = 20    # 0...ziemlich gross: Verteilungsbreite der Winkel
def get_xhyh(n,dt):
    phi_rand = random.vonmisesvariate(mu , kappa)
    return [(n*dt)* x for x in [math.cos(phi_rand), math.sin(phi_rand)]]

# Setzen der Anfangsparameter
dt = 0.0000001
```

```
19 eps = 0.0000001
20 n = 3/4
21
22 # Für graphische Ausgabe
23 plt.ion()
24 fig, ax = plt.subplots()
25 xp, yp = [], []
26 xph, yph = [], []
27 sc1 = ax.scatter(xph, yph, marker='^')
28 sc2 = ax.scatter(xp, yp, c="red")
29 plt.xlim(0, 1.2)
30 plt.ylim(0, round(n / (1 - n ** 2) + 0.5))
31 plt.title('Bouguer-Kurve (enhanced)')
32 fig.canvas.manager.set_window_title('SVLFG Aufgabe 03 (enhanced)')
33 #fig.canvas.set_window_title('SVLFG Aufgabe 03 (enhanced)')
34 plt.text(0.15, 0.9 * round(n / (1 - n ** 2) + 0.5), r'$n=$' + '{:3.2f}   '.format(n) +
35         r'$\Delta t=$' + '{:9.8f}   '.format(dt) + r'$\epsilon=$' + '{:9.8f}'.format(eps))
36 plt.ylabel('Entfernung')
37 plt.xlabel('Entfernung')
38 # plt.grid(True)
39 plt.draw()
40 ###################################
41
42 start = time.time()
43 t = 0.
44 xh = 1.
45 yh = 0.
46 x = 0.
47 y = 0.
48 k = 0.
49 phi = 0.
50 dx = dt
51 dy = 0.
52 dxh = 0.
53 dyh = 0.
54 while (math.fabs(y - yh) > eps or math.fabs(x - xh) > eps) and k < 10000000000:
55     k = k + 1
56     t = t + dt
57     x = x + dx
58     y = y + dy
59     xh = xh + dxh
60     yh = yh + dyh
61     if math.fabs(xh - x) != 0:
62         phi = math.atan2((yh - y) , (xh - x)) # da es in alle Richtungen gehen kann: atan2
63     else:
64         phi = math.pi / 2
65     dx = dt * math.cos(phi)
66     dy = dt * math.sin(phi)
67     if math.fmod(k, 100000) == 0:
68         dxh, dyh = get_xhyh(n,dt) # damit die Richtung auch sichtbar wird....
69         # print("phi=", '{:20.18f}'.format(phi), " xh-x =",
70         #       '{:20.18f}'.format(math.fabs(x - xh)),
71         #       "    yh-y =", '{:20.18f}'.format(math.fabs(y - yh)))
72         # print("Hyp=", '{:20.18f}'.format(math.sqrt((yh-yy)**2+(1-xx)**2)) ,
73         #       "     xh-xx =", '{:20.18f}'.format(math.fabs(xx - xh)),
74         #       "   yh-yy =", '{:20.18f}'.format(math.fabs(yy - yh)))
75         # Graphische Ausgabe
76         xp.append(x)
77         yp.append(y)
78         xph.append(xh)
79         yph.append(yh)
```

```
80          sc1.set_offsets(np.c_[xph, yph])
81          sc2.set_offsets(np.c_[xp, yp])
82          fig.canvas.draw_idle()
83          plt.pause(0.001)
84  distance = yh
85  plt.plot([0, xh], [distance, distance], 'b')
86  plt.text(xh, distance, '{:9.8f}'.format(distance))
87
88  #plt.draw()
89  fig.canvas.draw_idle()
90  ########################
91  print(dt, "/", eps, ":", "k0=", k, distance, n / (1 - n ** 2))
92  print("xh-x=", math.fabs(x - xh), "␣␣␣yh-y=", math.fabs(y - yh), "␣␣eps=", eps)
93  ende = time.time()
94  print('{:5.3f}s'.format(ende - start))
95  plt.waitforbuttonpress()
```